Universitat de Lleida

# TREBALL FINAL DE GRAU


Estudiant: David Jovellar Fantova

Titulació: Grau en Enginyeria Informàtica

Títol de Treball Final de Grau: **Aplicación web para concienciar sobre los factores implicados en la videoadicción**

Director/a: **Roberto García González**
**Rosa María Gil Iranzo**

Presentació

Mes: Gener

Any: 2021

# Índice

# Índice de figuras

# Índice de tablas

# 1. Introducción

El trabajo final de grado (TFG) que se describe en este documento trata sobre el desarrollo de una aplicación web que permite, de forma interactiva, concienciar sobre los factores y las consecuencias relacionadas con la adicción en los videojuegos (videoadicción).

De forma resumida, esto se intenta conseguir mediante la agrupación de representaciones virtuales de estos factores y consecuencias según la valoración del usuario que use la aplicación web.

Para conseguir este grado de interactividad se ha usado el Framework de desarrollo de aplicaciones Web Angular, debido a que se trata de una tecnología que dota al desarrollador de posibilidades interactivas muy potentes.

## 1.1 Motivación

Llegado el momento de decir la temática y las tecnologías implicadas para realizar el TFG tan solo tenía dos cosas claras:

- Desarrollar una aplicación web mediante el Framework Angular, ya que era una tecnología con la que había trabajado anteriormente en la Universidad y sentía curiosidad por seguir profundizando en ella.
- Desarrollar una aplicación web que pudiera ser aprovechada una vez finalizado el desarrollo, es decir, que aportará valor a la sociedad.

Este proyecto cumple ambos puntos, por lo que es lo que más me ha motivado para desarrollarlo.

## 1.2 Objetivos

El objetivo principal de este proyecto es desarrollar una aplicación interactiva que ayude y facilite la comprensión del proceso de videoadicción en las personas (Gil & Arnedo-moreno, 2020).

Esta comprensión se busca conseguir mediante los siguientes objetivos secundarios (Gil & Arnedo-moreno, 2020):

- La concienciación en las personas de los diseños que hay detrás de los videojuegos para engancharlos a jugar, para que sigan consumiendo sus productos aunque dejen de ser divertidos en sí mismos.
- Que las personas sean capaces de utilizar esta concienciación para evaluar críticamente estos diseños y sus riesgos, y que puedan autocontrolar su tiempo de juego en consecuencia.

Todo esto se pretende conseguir mediante la aplicación interactiva que se va a desarrollar, para enfatizar que los juegos no son una cosa mala en sí misma, sino que depende del uso que se les dé y de cómo se utilicen (Gil & Arnedo-moreno, 2020).

## 1.3 Estructura del resto del documento

El resto del documento se divide en varias secciones encargadas de detallar el estado del arte, toda la información relativa al desarrollo, las conclusiones del trabajo realizado, el trabajo futuro a realizar y la bibliografía.

En primer lugar, se puede encontrar la sección de estado del arte. Esta sección se encarga de presentar y explicar en qué consiste las distintas tecnologías utilizadas durante todo el desarrollo de la aplicación.

En segundo lugar, encontramos la sección relativa a todo el proceso de desarrollo de la aplicación. En esta sección se detalla la metodología seguida durante el desarrollo, la planificación temporal final y las distintas iteraciones que componen todo el desarrollo. Por cada iteración se detallan los requerimientos solicitados, el diseño, la implementación realizada y las conclusiones de la iteración concreta.

En tercer lugar encontramos la sección relativa a las conclusiones globales de todo el trabajo realizado. En esta sección se analizan todo lo que ha supuesto este trabajo personalmente y se da una opinión sobre él.

En cuarto lugar, encontramos la sección de trabajo futuro. En esta sección se proponen posibles ideas a implementar sobre la aplicación desarrollada con el objetivo de sumar funcionalidades y aportar valor a la aplicación.

Finalmente, la última sección del documento se encarga de indicar todas las fuentes consultadas tanto para desarrollar la aplicación como para realizar este documento.

# 2. Estado del arte

En esta sección se van a detallar las principales tecnologías involucradas en el desarrollo de todo el proyecto.

## 2.1 Typescript

Typescript es un lenguaje de programación desarrollado y mantenido por Microsoft. Se trata de un superset de JavaScript, esto quiere decir que, además de todas las ventajas que ofrece un lenguaje mundialmente reconocido en el sector como es JavaScript, añade nuevas funcionalidades que lo alzan como una gran oportunidad para desarrollar proyectos web (*TypeScript: Typed JavaScript at Any Scale.*, 2021).

Entre estas nuevas funcionalidades de Typescript sobre JavaScript se pueden destacar (Colina, 2019):

- Verificación de tipo de dato (Tipado).
- Soporte de las distintas versiones de EcmaScript.
- Programación orientada a objetos más sencilla.
- Lenguaje compilado, evita errores en la ejecución.
- Facilita la legibilidad en proyectos de gran envergadura.

## 2.2 Single Page Application (SPA)

La principal característica de una aplicación SPA, tal como su propio nombre indica, consiste en que todo el contenido está alojado en una sola página, es decir, todo el contenido se encuentra dentro de un único fichero HTML (Palomares, 2019).

Por lo tanto, todos los archivos HTML, CSS y JavaScript se cargan únicamente al inicio y es el Cliente (Navegador) el encargado de realizar la navegación entre las distintas páginas de la aplicación (Palomares, 2019).

Las ventajas de utilizar una página SPA son las siguientes (Rodríguez Patiño, 2021):

- Agiliza la lógica de negocio al procesarse toda ella en el lado del cliente.
- Agiliza la navegación ya que todo el contenido inicial (Css, JavaScript, imágenes…) se carga de golpe. Por lo tanto, incrementa la fluidez y mejora la experiencia de usuario.
- Ayuda a abaratar los costos de servidor al disminuir el número de peticiones realizadas.

Como en toda tecnología, también encontramos una serie de desventajas (Rodríguez Patiño, 2021):

- Dificulta el trabajo a los motores de búsqueda ya que no suelen ser capaces de leer o interpretar contenido cargado en segundo plano.
- La primera carga puede ser costosa debido a la cantidad de recursos que el navegador debe cargar.
- Al disponer de la lógica en el lado del cliente, es necesario aumentar la seguridad.

## 2.3 Bootstrap

Bootstrap se trata de un conjunto de herramientas predefinidas para desarrollar aplicaciones web responsivas con HTML, CSS y JavaScript (Suárez, 2020).

Consiste en un conjunto de reglas prediseñadas de CSS y JavaScript que te proveen de maquetaciones personalizables para utilizar en tu sitio web. Entre las más populares podemos encontrar distintos diseños de Navbars, Tablas, Botones, Formularios, Paginación etc. (Bootstrap, 2021a).

Además, una de las características más importantes de Bootstrap la encontramos en su Grid System. Gracias a este sistema de filas y columnas permite estructurar, de forma responsiva, todo el contenido con suma facilidad (Bootstrap, 2021c).

Este Framework se ha utilizado en el proyecto como complemento para facilitar el trabajo de maquetación y estructuración responsiva de los distintos componentes de cada pantalla (Bootstrap, 2021b).

## 2.4 Angular

El Framework Angular es la tecnología principal utilizada para desarrollar la aplicación. El resto de las tecnologías descritas anteriormente forman parte o complementan este Framework (Quality devs, 2019).

Desarrollado por Google, de código abierto y publicado en 2016, este Framework ha ido ganando popularidad con los años hasta convertirse en un referente dentro del desarrollo de Aplicaciones Web. Su funcionamiento se basa en la filosofía Single Page Application (SPA) y utiliza Typescript como lenguaje base (Quality devs, 2019).

Además de las ventajas, descritas anteriormente, que le provee basarse en la filosofía SPA y utilizar el lenguaje Typescript, son varias las funcionalidades que han hecho este Framework tan popular, entre ellas podemos destacar (Quality devs, 2019):

- Separación de la parte Frontend y la parte Backend, interconectándolos habitualmente mediante peticiones HTTP.
- Evita la duplicidad de código sin añadir complejidad.
- Implementa el Patrón Modelo-Vista-Controlador (MVC), el cual permite mantener el código mucho más ordenado facilitando el mantenimiento y la escalabilidad del código.

# 3. Desarrollo

## 3.1 Metodología

La metodología seguida durante el desarrollo ha consistido en dividir los requerimientos a realizar en iteraciones. Estas iteraciones permiten segmentar el desarrollo y mejorar la coordinación entre los requerimientos y la implementación.

Respecto al ámbito técnico del desarrollo, se ha creado un repositorio público en GitHub (https://github.com/DJovellar/Videoadiccio) para compartir en todo momento el código que se ha ido desarrollando. En este repositorio se creó una rama llamada Master encargada de almacenar el código relativo a Producción y una rama llamada Develop donde ir implementando las distintas iteraciones del desarrollo sin afectar al código de Producción (Atlassian, 2021).

Finalmente, se ha utilizado GitHub Pages para desplegar la aplicación en un servidor público y poder poner en práctica la aplicación a medida que se iban desarrollando. Este servidor se puede encontrar en https://djovellar.github.io/Videoadiccio/home.

## 3.2 Planificación Temporal

En la siguiente tabla se puede observar la planificación final del proyecto.

| **Tarea Realizada** | **Fecha Inicio** | **Fecha Final** | **Horas** |
|---|---|---|---|
| Familiarización con las tecnologías a utilizar | 1 de Septiembre de 2020 | 18 de Septiembre de 2020 | 72 horas |
| Análisis general de requerimientos | 21 de Septiembre de 2020 | 23 de Septiembre de 2020 | 12 horas |
| Desarrollo | 24 de Septiembre de 2020 | 5 de Diciembre de 2020 | 328 horas |
| Despliegue y Testeo global de la aplicación | 7 de Diciembre de 2020 | 11 de Diciembre de 2020 | 20 horas |
| Documentación | 14 de Diciembre de 2020 | 5 de Enero de 2021 | 108 horas |
| **TOTAL** | | | **540 horas** |

*Tabla 1. Planificación final*

De la anterior tabla podemos sacar varias conclusiones, empezando por el hecho de que fue necesario invertir una cantidad relativamente grande de horas para familiarizarse y comprender mejor el funcionamiento de las tecnologías a utilizar en el posterior desarrollo de la aplicación. Estas tecnologías están detalladas en la sección **Estado del Arte**.

Tras tener bien interiorizado el conocimiento de estas tecnologías, se realizó una reunión para analizar los requerimientos globales de la aplicación a desarrollar. Esta reunión se centró en detallar el objetivo que se buscaba al desarrollar esta aplicación y en cómo debería ser su funcionamiento.

A continuación se comenzó con el desarrollo de la aplicación, donde se puede observar que se destinan el grueso de las horas de toda la planificación.

En la siguiente tabla se puede observar el desglose temporal por iteraciones de la tarea de Desarrollo.

| Iteración | Fecha Inicio | Fecha Final | Horas |
|---|---|---|---|
| Iteración 1 | 24 de Septiembre de 2020 | 1 de Octubre de 2020 | 40 horas |
| Iteración 2 | 6 de Octubre de 2020 | 18 de Octubre de 2020 | 64 horas |
| Iteración 3 | 22 de Octubre de 2020 | 2 de Noviembre de 2020 | 60 horas |
| Iteración 4 | 3 de Noviembre de 2020 | 5 de Noviembre de 2020 | 12 horas |
| Iteración 5 | 6 de Noviembre de 2020 | 9 de Noviembre de 2020 | 24 horas |
| Iteración 6 | 10 de Noviembre de 2020 | 16 de Noviembre de 2020 | 36 horas |
| Iteración 7 | 18 de Noviembre de 2020 | 28 de Noviembre de 2020 | 56 horas |
| Iteración 8 | 29 de Noviembre de 2020 | 5 de Diciembre de 2020 | 36 horas |
| **TOTAL** | | | **328 horas** |

*Tabla 2. Planificación de las iteraciones*

Tras terminar con el desarrollo de la aplicación, esta ha quedado desplegada y disponible públicamente utilizando la herramienta gratuita GitHub Pages.

Finalmente, el resto de las horas se destinaron a documentar toda la información relativa a la aplicación y a su desarrollo.

## 3.3 Presupuesto

Para calcular el presupuesto del proyecto no se ha tenido en cuenta la tarea '**Familiarización con las tecnologías a utilizar**', ya que en un desarrollo real se presupone que ya se debería tener el conocimiento necesario para desarrollar el proyecto contratado.

Además, se han tenido en cuenta los gastos indirectos del entorno de trabajo. Entre este tipo de gastos nos podemos encontrar la electricidad, el equipo de trabajo o el hosting.

Finalmente, basándonos en la tendencia actual del mercado para desarrolladores freelance con baja experiencia, se ha fijado un precio base por hora de 12 euros.

En la siguiente tabla se puede observar el desglose del presupuesto total del proyecto.

| Concepto | Tipo | Horas | Precio |
|---|---|---|---|
| Análisis general de requerimientos | Variable | 12 horas | 144 Euros |
| Desarrollo | Variable | 328 horas | 3.936 Euros |
| Despliegue y Testeo global de la aplicación | Variable | 20 horas | 240 Euros |
| Documentación | Variable | 108 horas | 2.160 Euros |
| Hosting | Fijo | X | Gratuito |
| Electricidad | Fijo | X | 80 Euros |
| **TOTAL** | | | **6.560 Euros** |

*Tabla 3. Presupuesto del proyecto*

## 3.4 Iteraciones

### 3.4.1 Iteración 1

#### *3.4.1.1 Requerimientos / Especificación*

***Página de inicio***

Diseñar una página de bienvenida al acceder a la aplicación, esta página debe contener el formulario para crear una sesión y una descripción sobre de qué trata esta aplicación.

***Guardar información que forman las tarjetas***

Las tarjetas se crearán a partir de información predefinida que se debe obtener desde la propia aplicación. Esta información, por el momento, será ficticia.

***Diseño de las tarjetas***

Diseñar la estructura visual de una tarjeta, que estará formada por una imagen, un nombre y una descripción.

***Página de sesión***

Diseñar la pantalla que representará la sesión, la cual estará formada por varias categorías ficticias entre las cuales el usuario podrá ir arrastrando las tarjetas, agrupándolas según su propio criterio.

### 3.4.1.2 Diseño


*Figura 1. Iteración 1 - Página Home*


*Figura 2. Iteración 1 – Página de sesión*


*Figura 3. Iteración 1 - Diseño de tarjeta*

### *3.4.1.3 Implementación*

***Página de inicio***

Para implementar la página de inicio se ha creado un componente llamado Home de tipo Page, el cual está formado por un formulario que permite introducir un nombre de usuario, una imagen y una descripción ficticia sobre la aplicación.

El formulario ha sido implementado mediante la versión de los formularios reactivos en Angular, los cuales permiten una mayor flexibilidad en el control de los campos (validaciones personalizadas, feedback visual de errores, etc.). Además, los formularios reactivos permiten definir la mayoría de código en la clase Controladora, lo cual facilita la escalabilidad y mantiene el código HTML mucho más limpio (Google, 2020a).

Inicialmente se ha definido como validación que el nombre de usuario no pueda estar vacío.

***Guardar información que forman las tarjetas***

La información de las tarjetas se ha guardado en un fichero de tipo JSON, donde cada tarjeta es un objeto que contiene la información correspondiente. Cada objeto está formado por dos atributos, el nombre y la descripción de la tarjeta.

```
{
    "name": "Tarjeta 1",
    "content": "Aqui se mostrara el texto descriptivo de la tarjeta 1
                , es decir, en que consiste en concreto"
},
{
    "name": "Tarjeta 2",
    "content": "Aqui se mostrara el texto descriptivo de la tarjeta 2
                , es decir, en que consiste en concreto"
},
```

***Diseño de las tarjetas***

Para realizar el diseño de las tarjetas se ha realizado una personalización del componente Card de Bootstrap. Además, se ha definido un componente de Angular encargado de representar una tarjeta.

```
<div class="card">
  <img class="card-img-top"
       src="../../../assets/img/portada.jpg"
       height="250" width="250" alt="Card image cap">
  <div class="card-body text-center">
    <h5 class="card-title">{{card.name}}</h5>
    <hr>
    <p class="card-text">{{card.content}}</p>
  </div>
</div>
```

Para obtener y cargar la información de cada tarjeta se ha definido una Clase que representa el objeto Tarjeta y un Servicio encargado de leer el fichero JSON que contiene dicha información. Una vez leída la información, el Servicio proporciona esta información al Controlador de esta vista para que pueda mostrarla.

```
  getAllCards(): Observable<Card[]> {
    return this.http.get<Card[]>('assets/cards/data.json');
  }
```

***Página de sesión***

Para implementar la agrupación entre categorías se ha creado el componente Session de tipo Page, el cual usa el módulo DragDrop introducido en Angular 7. Este módulo proporciona herramientas que permiten convertir las tarjetas en elementos arrastrables. Además, permite definir contenedores entre los cuales arrastrar las tarjetas y ordenarlas según la posición en la que se arrastre el elemento (Google, 2020c).

Como contenedores se han definido 4 listas de tarjetas encargadas de guardar las tarjetas contenidas en los contenedores.

```
    <div class="row mt-3">
        <div class="col-4"></div>
        <div class="col-3 col-container">
            <h5 class="text-center">Pendiente</h5>
            <hr>

            <div class="card-container"
                 cdkDropList
                 #pendingList="cdkDropList"
                 [cdkDropListConnectedTo]="[category1List, category2List,
 category3List]"
                 [cdkDropListData]="cards"
                 (cdkDropListDropped)="onDrop($event)">

                <app-
card *ngFor="let card of cards" [card]="card" cdkDrag></app-card>
            </div>
        </div>
    </div>
```

#### *3.4.1.4 Resultados / Conclusiones*

Esta primera iteración ha servido para definir la estructura base del proyecto e implementar la funcionalidad más básica de la aplicación.

Aunque la pantalla de Sesión pueda diferir del resultado final, gracias a ella se ha profundizado en la tecnología y permitirá en el futuro agilizar la implementación de los requerimientos reales.

### 3.4.2 Iteración 2

#### *3.4.2.1 Requerimientos / Especificación*

***Modificación del diseño de las tarjetas***

Rediseñar las tarjetas, las cuales estarán formadas por una imagen predefinida la cual ya contiene el nombre y la descripción de dicha tarjeta.

***Presentación deslizante de las tarjetas***

Diseñar un Slideshow deslizante, mostrando la tarjeta seleccionada a mayor tamaño y previsualizando a menor tamaño las tarjetas anterior y siguiente.

Este Slideshow mostrará el número de tarjetas totales y la posición actual de la tarjeta mostrada.

***<u>Agrupar selecciones de distinto tipo que forman una relación</u>***

Definir 3 tipos de tarjetas (Hook, Risk y Catalyst) que deberán agruparse para formar una relación. Cada tipo de tarjeta dispondrá de su propia presentación deslizante para poder seleccionarlas.

***<u>Mostrar progreso de la relación actual</u>***

Mostrar un historial con las tarjetas de cada tipo seleccionadas en la relación actual, añadiendo la posibilidad de hacer zoom a cada tarjeta facilitando la lectura en entornos con tamaño reducido.

### *3.4.2.2 Diseño*

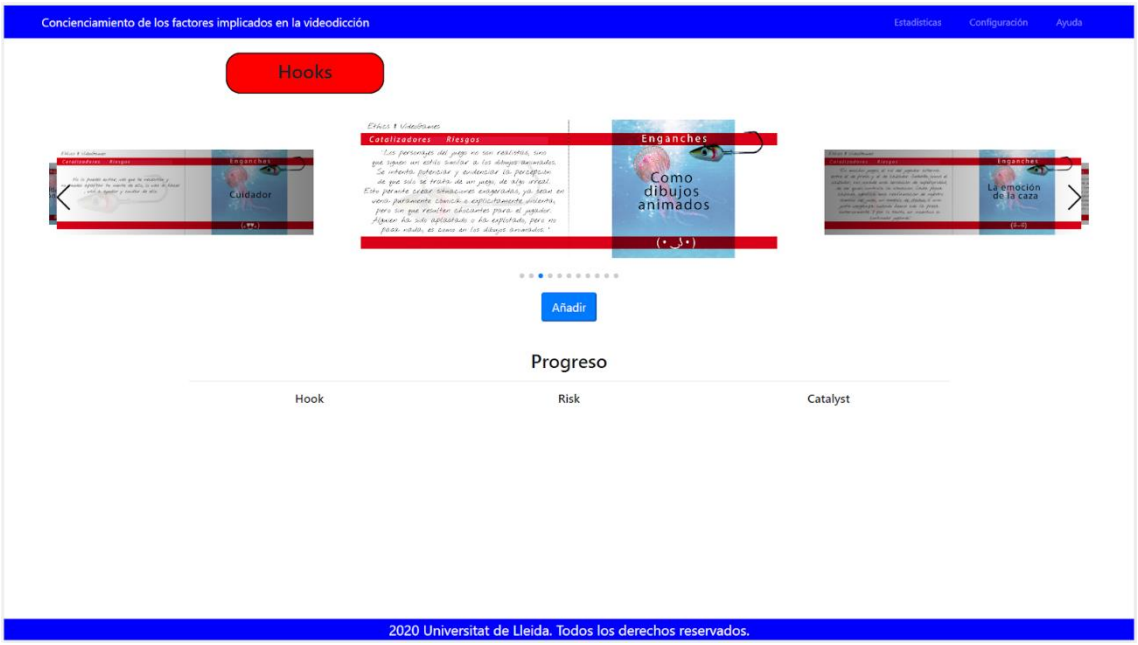


*Figura 4. Iteración 2 – Página de sesión*


*Figura 5. Iteración 2 - Añadir tarjeta Hook*

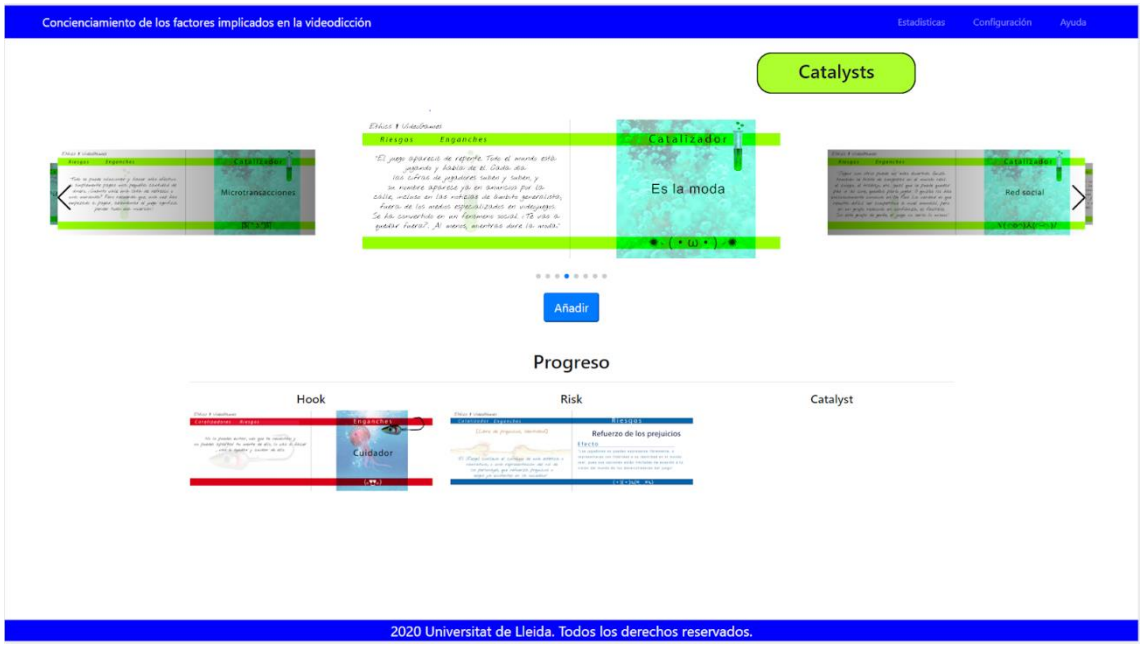


*Figura 6. Iteración 2 - Añadir tarjeta Risk*

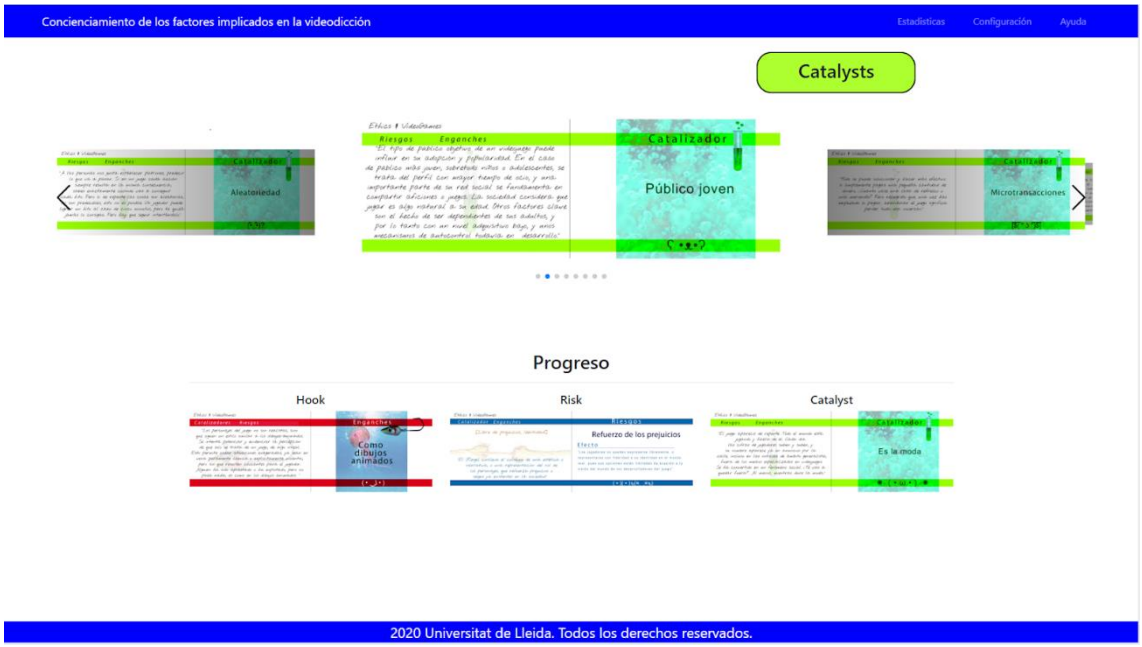


*Figura 7. Iteración 2 - Añadir tarjeta Catalyst*

### *3.4.2.3 Implementación*

***Modificación del diseño de las tarjetas***

Para disponer de las imágenes de cada tarjeta, éstas se han añadido en la carpeta de recursos del proyecto.

Posteriormente se ha modificado la estructura del fichero JSON que contiene la información de cada tarjeta, añadiendo el nombre real de cada tarjeta, sustituyendo la descripción por la ruta a la imagen correspondiente a cada tarjeta y añadiendo el tipo de tarjeta del que se trata.

```
{
    "name": "La vida no tiene boton de pausa",
    "content": "../../../assets/img/hooks/enganche_botonpausa.jpg",
    "type": "Hook"
},
{
    "name": "Cuidador",
    "content": "../../../assets/img/hooks/enganche_cuidador.jpg",
    "type": "Hook"
},
```

Finalmente, el diseño de las tarjetas se ha modificado sustituyendo el Componente Tarjeta por la etiqueta <img>, la cual carga la imagen predefinida de la tarjeta.

***Presentación deslizante de las tarjetas***

Para implementar la presentación de las tarjetas se ha creado un componente que utiliza la librería SwiperJS, la cual implementa un Slideshow con multitud de parámetros a configurar. Algunos de los parámetros utilizados son (Kharlampidi, 2020):

- **SlidesPerView**: Cantidad de tarjetas a mostrar en cada Slide.
- **Pagination**: Pagination para indicar la cantidad total de tarjetas y la posición concreta de la tarjeta mostrada.
- **Effect**: Se ha utilizado el efecto Coverflow, el cual permite aumentar la profundidad de las tarjetas adyacentes y sombrearlas.

```
configureSwiper() {
  this.mySwiper = new Swiper('.swiper-container', {
    loop: true,
    effect: 'coverflow',
    grabCursor: true,
    slidesPerView: 2,
    spaceBetween: 350,
    centeredSlides: true,
    observer: true,
    observeParents: true,
    parallax: true,
    coverflowEffect: {
      rotate: 0,
      stretch: 0,
      depth: 500,
      modifier: 1,
      slideShadows : true,
    },
    pagination: {
      el: '.swiper-pagination',
    }
  });
}
```

***Agrupar selecciones de distinto tipo que forman una relación***

Se ha utilizado el mismo componente que representa el Slideshow para mostrar la presentación de los 3 tipos de tarjetas. Esto se ha conseguido creando una variable con la lista de cartas a mostrar en el componente Slideshow, esta lista se la provee el componente Session. Además, se ha creado una etiqueta identificativa de cada tipo de tarjeta mostrando el Slideshow con su etiqueta correspondiente y ocultando el resto de las etiquetas.

```
    <div class="row pt-4">
        <div class="col-12">
            <app-slideshow [cards]="activeCards"
                           (selectedCard)="getCardSelected($event)"></app
-slideshow>
        </div>
    </div>
```

Para añadir una tarjeta a la relación actual, se ha definido un botón encargado de obtener el Slide mostrado al pulsar, enviando esta tarjeta al componente Session para que este actualice la relación actual. Posteriormente, el componente Session envía al componente Slideshow el siguiente grupo de tarjetas a mostrar.

```
  addCard() {
    const currentSlide = this.mySwiper.activeIndex;
    this.mySwiper.activeIndex = 0;
    this.selectedCard.emit({ card: this.cards[currentSlide] });

    if (this.cards[currentSlide].type === 'Catalyst') {
      this.showAdd = false;
    }
  }
```

Finalmente se ha creado una clase llamada RelationCards encargada de representar una relación. Esta clase está formada por 3 atributos de tipo Card, cada uno de ellos representando un tipo de tarjeta (Hook, Risk y Catalyst).

Al iniciar la sesión se crea un objeto de esta clase para ir guardando el progreso de la relación.

***Mostrar progreso de la relación actual***

Se ha creado un componente encargado de mostrar el progreso de la relación actual, en el cual se muestra la tarjeta seleccionada o un espacio vacío si todavía no se ha seleccionado la tarjeta de ese tipo.

```
<div class="row text-center">
    <div class="col-4">
        <h5>Hook</h5>
        <div class="zoom">
            <img [ngStyle]="{'opacity': disabledCards ? '0.2' : '1'}"
                 *ngIf="relationCards.hook"
                 [src]="relationCards.hook.content"
                 class="img-fluid">
        </div>
    </div>
    <div class="col-4">
        <h5>Risk</h5>
        <div class="zoom">
            <img [ngStyle]="{'opacity': disabledCards ? '0.2' : '1'}"
                 *ngIf="relationCards.risc"
                 [src]="relationCards.risc.content"
                 class="img-fluid">
        </div>
    </div>
    <div class="col-4">
        <h5>Catalyst</h5>
        <div class="zoom">
            <img [ngStyle]="{'opacity': disabledCards ? '0.2' : '1'}"
                 *ngIf="relationCards.catalyst"
                 [src]="relationCards.catalyst.content"
                 class="img-fluid">
        </div>
    </div>
</div>
```

Para poder mostrar esta información, el componente Session le pasa por parámetro el objeto RelationCards encargado de guardar el progreso de la relación.

```
<div class="container pt-5">
        <h2 class="text-center">Progreso</h2>
        <hr>
        <app-historical-session
            [relationCards]="relationCards"
            [disabledCards]="continue"></app-historical
session>
</div>
```

Además, se ha implementado un auto zoom al pasar el ratón por encima de cada tarjeta usando animaciones de CSS.

```
.zoom {
    transition: transform .3s;
}

.zoom:hover {
  -ms-transform: scale(1.8); /* IE 9 */
  -webkit-transform: scale(1.8); /* Safari and Chrome */
  transform: scale(1.8);
}
```

#### *3.4.2.4 Resultados / Conclusiones*

Tras esta iteración se ha conseguido la primera aproximación real de lo que puede acabar siendo la aplicación.

Se ha definido el núcleo de funcionalidad clave, el cual consiste en mostrar, agrupar y guardar conjuntos de tarjetas según su tipo. También se ha mejorado la presentación visual de la pantalla de sesión.

### 3.4.3 Iteración 3

#### *3.4.3.1 Requerimientos / Especificación*

**Guardar relación actual e iniciar una nueva relación**

Con los 3 tipos de tarjetas seleccionadas se habilitará un botón para guardar la relación y comenzar otra nueva.

Tras guardar la relación, se preguntará al usuario si quiere mantener alguna de las tarjetas guardadas en la siguiente relación.

**Mostrar histórico de relaciones realizadas**

Se creará una nueva pantalla, accesible durante la sesión en progreso, para consultar el histórico de relaciones realizadas hasta el momento.

En este histórico se mostrarán todas las relaciones realizadas, indicando las tarjetas que forman cada relación.

**Finalizar una sesión**

Habilitar un botón en la sesión que permite finalizarla. Dado que es una acción sumamente importante dentro de la sesión, se requerirá una confirmación por parte del usuario de esta finalización.

### 3.4.3.2 Diseño


*Figura 8. Iteración 3 - Guardar relación*

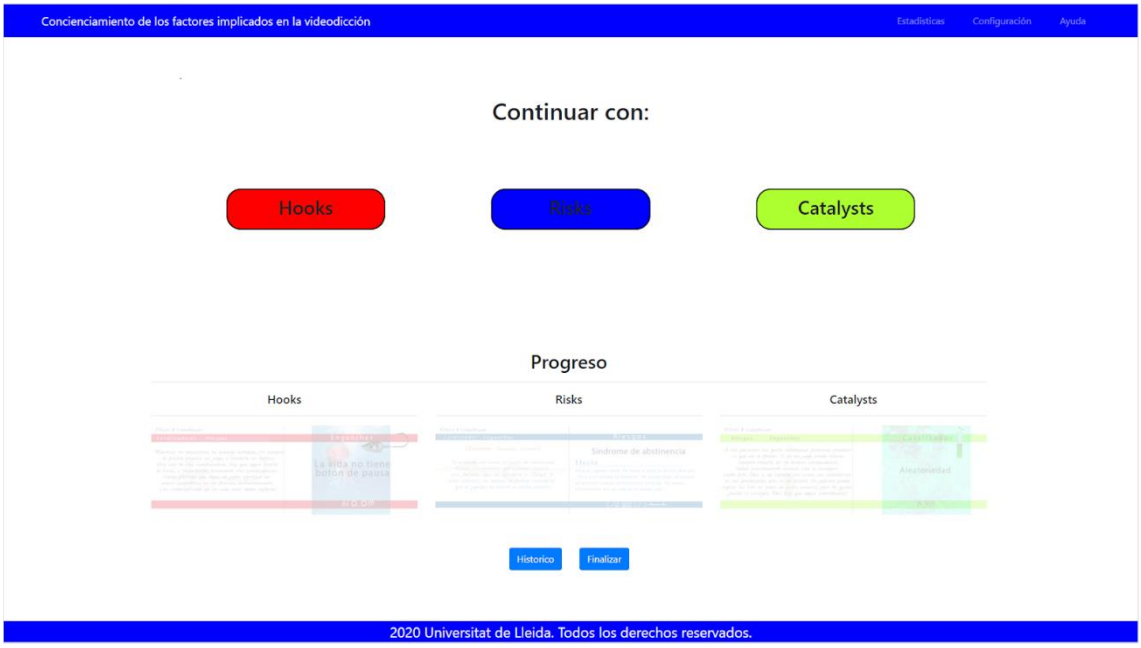


*Figura 9. Iteración 3 - Continuar relación guardada*


*Figura 10. Iteración 3 - Histórico de relaciones*

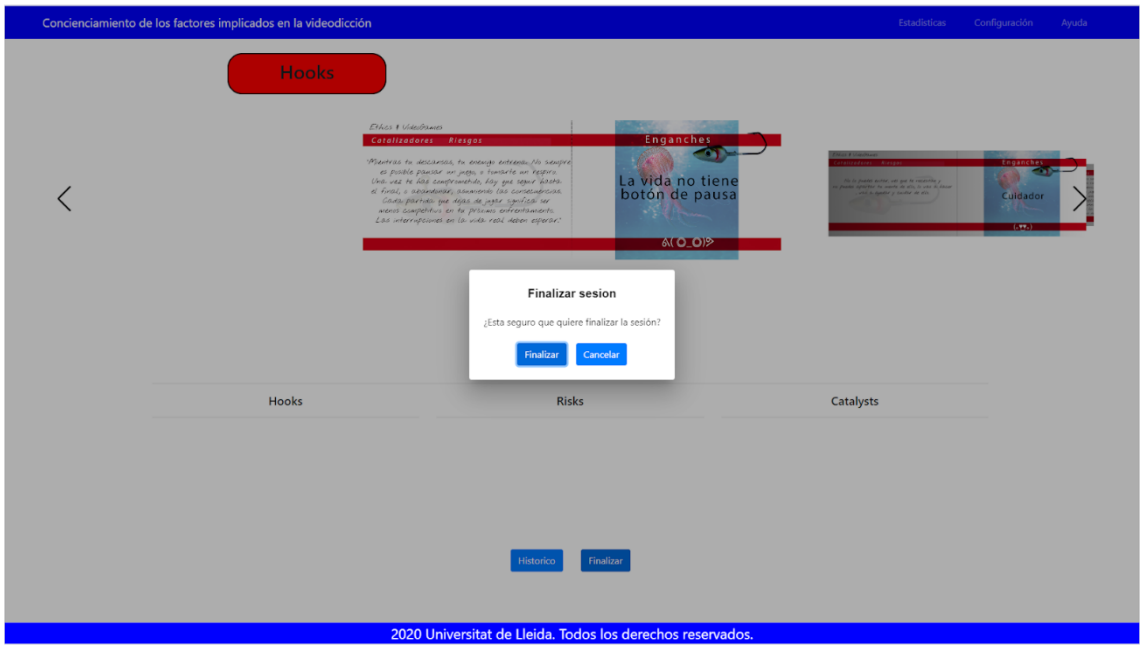


*Figura 11. Iteración 3 - Finalizar sesión*

### *3.4.3.3 Implementación*

***<u>Guardar relación actual e iniciar una nueva relación</u>***

Se ha definido un botón para guardar la relación que permanece oculto hasta que se completa el progreso de la relación actual.

También se ha creado un Servicio llamado Storage encargado de guardar el histórico de relaciones de la sesión, al pulsar el botón de guardar se añade el objeto de tipo RelationCards mediante este servicio a una lista que contiene todas las relaciones realizadas hasta el momento.

```
saveRelation( relation: RelationCards ) {
  this.relationsHistory.push(relation);
}
```

A continuación, se oculta el componente Slideshow y se muestran 3 botones que sirven para indicar en qué punto se quiere continuar la relación anterior.

Según el botón que se pulse se modifica el objeto de tipo RelationCards que se utiliza para guardar y mostrar las tarjetas de la relación actual:

- Hook: Se crea una nueva relación sin ninguna tarjeta definida.
- Risk: Se copia la tarjeta Hook de la relación anterior y se añade a la relación nueva.
- Catalyst: Se copian las tarjetas Hook y Risk y se añaden a la relación nueva.

De esta forma se consigue continuar con una nueva relación aprovechando tarjetas seleccionadas de la relación anterior.

```
nextIterationType(type: string) {
  const prevHook = this.relation.hook;
  const prevRisk = this.relation.risc;
  this.relation = new RelationCards();

  switch (type) {
    case 'Hook':
      this.relation.hook = this.defaultCard;
      this.typeCard = 'Hook';
      this.activeCards = this.hooks;
      break;
    case 'Risk':
      this.relation.hook = prevHook;
      this.typeCard = 'Risk';
      this.activeCards = this.risks;
      break;
    case 'Catalyst':
      this.relation.hook = prevHook;
      this.relation.risc = prevRisk;
      this.typeCard = 'Catalyst';
      this.activeCards = this.catalysts;
  }
  this.continue = false;
}
```

***Mostrar histórico de relaciones realizadas***

Se ha creado un nuevo componente de tipo Page encargado de representar la página que muestra el histórico de relaciones de la sesión.

El controlador de este componente utiliza el servicio Storage para obtener la lista de relaciones y se lo pasa a la vista para que las muestre en la pantalla.

```
  getRelationsHistory() {
    return this.relationsHistory;
  }
```

La vista de este controlador reutiliza el componente encargado de mostrar la relación actual en la página de sesión, realizando un bucle por cada relación y enviando al componente cada relación de la lista, para mantener la estructura visual de una relación y permitir el auto zoom sobre cada tarjeta.

```
<div class="container" *ngFor="let relation of history">
    <div class="pt-4"></div>
    <app-historical-
session [active]="relation"  [disabledCards]="false"></app-historical-
session>
    <hr>
</div>
```

Por último, se ha añadido un botón en la página de sesión para abrir el histórico de relaciones y un botón en la página del histórico para cerrarlo y volver a la página de la sesión.

***Finalizar una sesión***

Para permitir finalizar una sesión se ha añadido un botón, junto con el botón para mostrar el histórico ya existente, encargado de borrar el historial de relaciones y redirigir a la página principal.
De esta forma, se permite simular el inicio de una nueva sesión al no haber relaciones realizadas.

```
      dialogRef.afterClosed().subscribe(res => {
        if (res) {
          this.storageService.resetRelationsHistory();
          this.router.navigate(['home']);
        }
      });
```

Además, para implementar la ventana emergente que pide la confirmación de finalización se ha usado el módulo de Angular Mat Dialog, el cual permite lanzar ventanas emergentes definidas como un componente de Angular (Google, 2020b).

```
<h2 mat-dialog-title class="text-center">
    <b>{{data.title}}</b>
</h2>

<mat-dialog-content class="text-center"
                    style="white-space: pre;">
    {{data.message}}
</mat-dialog-content>

<div class="row text-center pt-4">
    <div class="col-12">
        <button class="btn btn-primary"
                [mat-dialog-close]="true">
            {{data.okButton}}
        </button>
        <span class="pr-3"></span>
        <button class="btn btn-primary"
                [mat-dialog-close]="false">
            {{data.noOkButton}}
        </button>
    </div>
</div>
```

El componente encargado de representar la ventana emergente se ha parametrizado, de tal forma que el componente padre decide qué información se mostrará en la ventana emergente. Gracias a esta parametrización, se podrá reutilizar este componente para lanzar ventanas emergentes personalizadas en cualquier parte de la aplicación.

```
      const dialogRef = this.dialog.open(DialogBoxComponent, {
      disableClose: true,
      data: {
        title: 'Finalizar sesion',
        message: '¿Esta seguro que quiere finalizar la sesión?',
        okButton: 'Finalizar',
        noOkButton: 'Cancelar'
      }
      });
```

#### *3.4.3.4 Resultados / Conclusiones*

Tras esta iteración se ha conseguido implementar una aproximación bastante completa del flujo de trabajo de una sesión, ya es posible agrupar y formar tantas relaciones como se crea conveniente, consultarlas durante de la sesión y terminar finalizando la sesión.

Aunque aún hay margen con relación a posibles nuevas funcionalidades en la sesión, se han definido las funcionalidades más importantes, lo que permite que se pueda realizar una sesión completa.

### 3.4.4 Iteración 4

#### *3.4.4.1 Requerimientos / Especificación*

***Guardar sesiones sin finalizar***

Mientras se está realizando una sesión, si el usuario sale de la pantalla de la sesión antes de finalizar deberá guardarse el progreso actual de la sesión.

***Reanudar una sesión sin finalizar***

Al acceder a la página principal se avisará al usuario de que hay una sesión sin finalizar y se le dará la opción de reanudarla o eliminarla.

Si no existe ninguna sesión sin finalizar, el funcionamiento de la pantalla inicial será el mismo.

#### *3.4.4.2 Diseño*

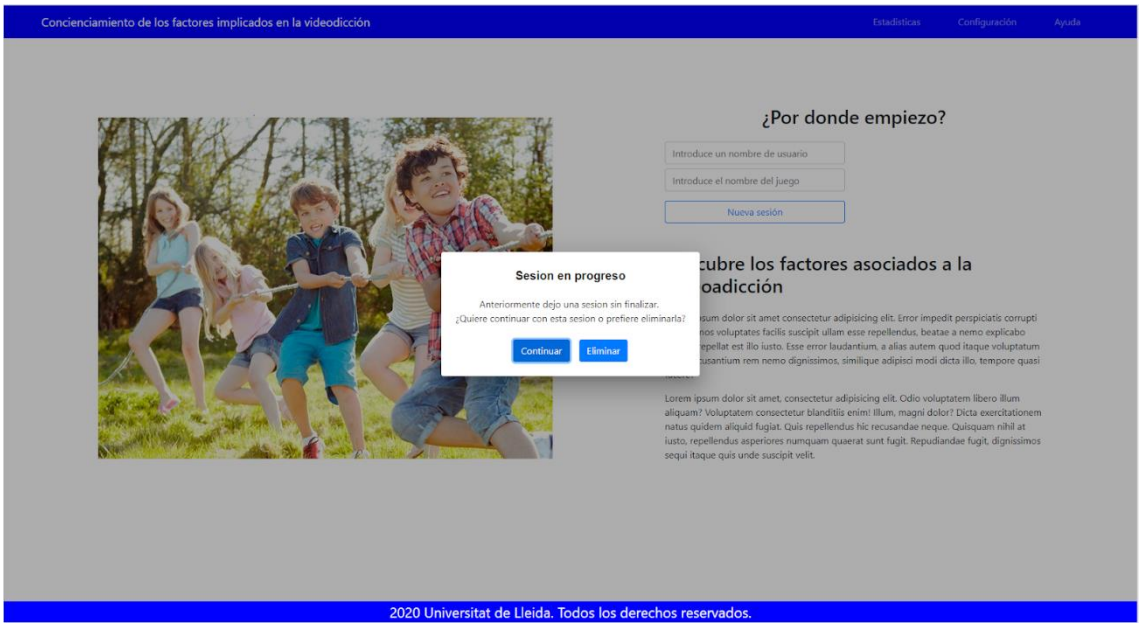


*Figura 12. Iteración 4 - Reanudar sesión*

#### *3.4.4.3 Implementación*

***Guardar sesiones sin finalizar***

Para mantener las sesiones sin finalizar guardadas se ha utilizado la caché del navegador como base de datos. La caché se trata de una pequeña base de datos, que disponen todos los navegadores actuales, basada en el paradigma Clave-Valor.

En Angular esta base de datos se conoce como Localstorage y permite guardar Strings asignándoles una clave identificativa. Por lo tanto, se ha definido un método en el servicio Storage encargado de guardar el historial de relaciones en esta base de datos.

```
saveRelation( relation: RelationCards ) {
  this.relationsHistory.push(relation);
  localStorage.setItem('historyRelations',
                          JSON.stringify(this.relationsHistory));
}
```

***Reanudar una sesión sin finalizar***

Al acceder a la aplicación, se carga la página Home y se consulta si en la caché hay algún histórico de relaciones guardado.

Para implementar esta funcionalidad se ha utilizado el Servicio Storage, declarando un método que consulta la caché del navegador y devuelve el histórico de la sesión si existe. Este método es llamado en el constructor del componente que representa la página Home.

```
getRelationsHistory() {
  this.relationsHistory = JSON.parse(
                             localStorage.getItem('historyRelations'));
  return this.relationsHistory;
}
```

Tras cargar la página Home, si el Servicio Storage detecta algún histórico se carga una ventana emergente informando de que existe una sesión sin finalizar. Este aviso ofrece la opción de reanudar la sesión o eliminar la sesión.

Si se elige reanudar la sesión, se crea una nueva sesión cargando previamente el histórico obtenido de la caché. En cambio, si se elige eliminar la sesión se borra el histórico de la caché mediante un método definido en el Servicio Storage.

```
dialogRef.afterClosed().subscribe(res => {
   if (res) {
     this.router.navigate(['session']);
   } else {
     this.storageService.resetRelationsHistory();
   }
 });
```

Para implementar la ventana emergente se ha utilizado el componente parametrizable implementado en la iteración anterior, cambiando el texto a mostrar y las acciones de los botones.

```
 const dialogRef = this.dialog.open(DialogBoxComponent, {
   disableClose: true,
   data: {
     title: 'Sesion en progreso',
     message: 'Anteriormente dejo una sesion sin finalizar. \n¿Quiere
               continuar con esta sesion o prefiere eliminarla?',
     okButton: 'Continuar',
     noOkButton: 'Eliminar'
   }
 });
```

#### *3.4.4.4 Resultados / Conclusiones*

Tras esta iteración se ha conseguido que la aplicación guarde las sesiones sin finalizar, permitiendo realizar la sesión sin necesidad de comenzar y terminando de forma ininterrumpida.

Esta funcionalidad puede ser muy importante, ya que hay muchas tarjetas de cada tipo lo que da la posibilidad de realizar una gran cantidad de relaciones. Esto provoca que una sesión pueda alargarse mucho y volverse tedioso tener que realizarla de golpe.

Además, se ha implementado la base que puede permitir aumentar la funcionalidad de guardado de sesiones en el futuro si fuera necesario.

### 3.4.5 Iteración 5

#### *3.4.5.1 Requerimientos / Especificación*

***Seleccionar grupo de tarjetas a visualizar***

Se utilizarán las cajas que indican que grupo de tarjetas se están mostrando (Hook, Risk, Catalyst) en el Slideshow como botones que servirán para elegir el grupo de tarjetas a mostrar en el Slideshow.

Si se selecciona un grupo de tarjetas y ya existe una tarjeta seleccionada para ese tipo en la relación actual, se eliminará esta selección.

***Añadir juego al crear sesión y mostrar esta información durante la sesión***

Se añadirá un nuevo campo para indicar el juego sobre el que se realizará la sesión al crearla en la página de inicio. Este nuevo campo debe ser obligatorio al crear la sesión.

Además, mientras se realiza la sesión se deberán visualizar en la cabecera el usuario y el juego correspondiente a esa sesión.

#### *3.4.5.2 Diseño*

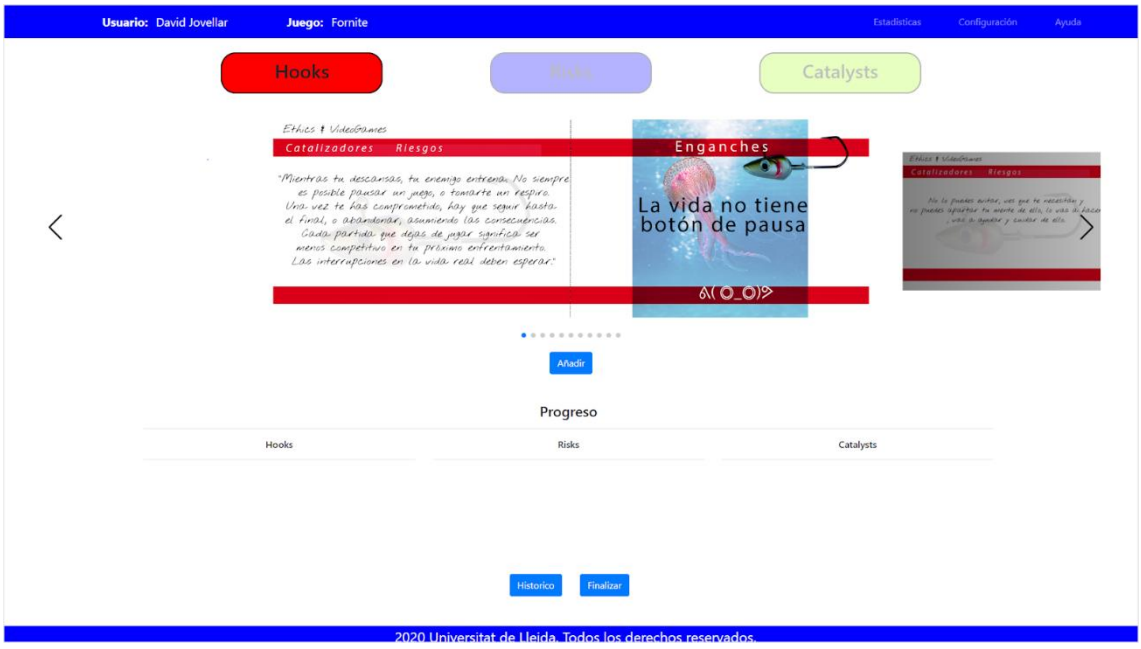


*Figura 13. Iteración 5 - Información de cabecera en Session*


*Figura 14. Iteración 5 - Información de cabecera en Histórico de Relaciones*

### *3.4.5.3 Implementación*

***Seleccionar grupo de tarjetas a visualizar***

Se han convertido las cajas que muestran cada tipo de tarjetas en botones que llaman a la función ya implementada que sirve para cambiar el tipo de tarjetas a mostrar al añadir una tarjeta. Mediante un argumento de la función se indica el grupo de tarjetas a cargar.

Para descartar la tarjeta o tarjetas se modifica a NULL el valor del atributo correspondiente a ese tipo de tarjeta.

Aprovechando la implementación anterior, se ha permitido visualizar tipos de tarjetas futuras sin haber añadido la tarjeta actual.

Para mantener el orden de selección, se comprueba si los tipos de tarjetas anteriores han sido seleccionados ocultando el botón de Añadir, mediante una variable booleana y la etiqueta *nglf, si alguno no ha sido seleccionado.

```
nextIterationType(type: string) {
  this.showAdd = true;
  this.showSave = false;

  const prevHook = this.relation.hook;
  const prevRisk = this.relation.risc;
  this.relation = new RelationCards();

  switch (type) {
    case 'Hook':
      this.relation.hook = this.defaultCard;
      this.typeCard = 'Hook';
      this.activeCards = this.hooks;
      break;
    case 'Risk':
      if (prevHook.name === 'default') {
        this.showAdd = false;
      }

      this.relation.hook = prevHook;
      this.typeCard = 'Risk';
      this.activeCards = this.risks;
      break;
    case 'Catalyst':
      if (prevHook.name === 'default' || !prevRisk) {
        this.showAdd = false;
      }

      this.relation.hook = prevHook;
      this.relation.risc = prevRisk;
      this.typeCard = 'Catalyst';
      this.activeCards = this.catalysts;
  }
  this.continue = false;
}
```

***Añadir juego al crear sesión y mostrar esta información durante la sesión***

Para añadir la opción de introducir un juego al crear la sesión se ha añadido otro campo llamado Game al formulario Reactivo que controla la creación de la sesión. A este nuevo campo se le han definido las mismas validaciones que ya dispone el campo Usuario.

Para controlar qué información mostrar en la cabecera (nombre de la aplicación o información de la sesión) se ha utilizado una variable Booleana que se le pasa al Componente Navbar. Según el valor de esta variable, se muestra el nombre o la información de la sesión. Además, tanto el Componente Sesión como el Componente History Relations le pasan la información de la sesión al Componente Navbar para que pueda mostrarlo en la pantalla.

```
<nav class="navbar navbar-expand-xl navbar-dark color-primary">

  <div *ngIf="isTitle" class="navbar-brand pl-5">
    {{message}}
  </div>

  <div *ngIf="!isTitle" class="navbar-brand"
       style="padding-left: 150px;">
    <span class="font-weight-bold">Usuario: </span>
    <span class="pl-2">{{user}}</span>
    <span style="padding-left: 100px;"></span>
    <span class="font-weight-bold">Juego: </span>
    <span class="pl-2">{{game}}</span>
  </div>

  <button class="navbar-toggler"
          type="button"
          data-toggle="collapse"
          data-target="#navbarNavAltMarkup"
          aria-controls="navbarNavAltMarkup"
          aria-expanded="false"
          aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse text-
center" id="navbarNavAltMarkup">
    <div class="navbar-nav ml-auto mr-5">
      <a class="nav-item nav-link mx-4" href="#">Estadisticas</a>
      <a class="nav-item nav-link mx-4" href="#">Configuración</a>
      <a class="nav-item nav-link mx-4" href="#">Ayuda</a>
    </div>
  </div>
</nav>
```

```
<app-navbar [isTitle]="false" [user]="user" [game]="game"></app-navbar>
```

```
<app-navbar [message]="title" [isTitle]="true"></app-navbar>
```

#### *3.4.5.4 Resultados / Conclusiones*

Tras esta iteración, se ha mejorado la funcionalidad relacionada con la realización de la propia sesión. Esto es logrado gracias a permitir descartar selecciones de tarjetas realizadas y pudiendo visualizar tarjetas de tipos futuros para ayudar a elegir el tipo de tarjeta actual.

Además, se ha añadido información que aumenta el detalle de la sesión en curso y permite visualizar esta información durante todo el progreso de la sesión.

### 3.4.6 Iteración 6

#### *3.4.6.1 Requerimientos / Especificación*

***Guardar múltiples sesiones en memoria simultáneamente***

Mejorar la funcionalidad encargada de guardar una sesión sin finalizar, guardando toda la información relacionada con la sesión en memoria (usuario, juego y relaciones realizadas). Además, se guardan todas las sesiones realizadas simultáneamente, independientemente de si se han finalizado o no.

***Página para mostrar las sesiones guardadas***

Se creará una nueva pantalla, accesible desde la cabecera de la aplicación, encargada de mostrar todas las sesiones guardadas en memoria.

Por un lado se mostrarán las sesiones pendientes de finalizar, permitiendo reanudarlas o eliminarlas. Por otro lado se mostrarán las sesiones finalizadas, permitiendo consultar el histórico de relaciones o eliminarlas.

Finalmente, al intentar eliminar una sesión se pedirá confirmación al usuario.

***Control de sesiones duplicadas***

Al crear una nueva sesión se comprobará que no exista ya una sesión, ya sea pendiente de finalizar o finalizada, con el usuario y el juego introducidos.

Si la sesión ya existe y está sin finalizar, se mostrará una ventana emergente preguntando al usuario si quiere reanudarla. En cambio, si la sesión ya existe pero está finalizada, se mostrará una ventana emergente informando al usuario.

### 3.4.6.2 Diseño


*Figura 15. Iteración 6 - Página de sesiones guardadas*

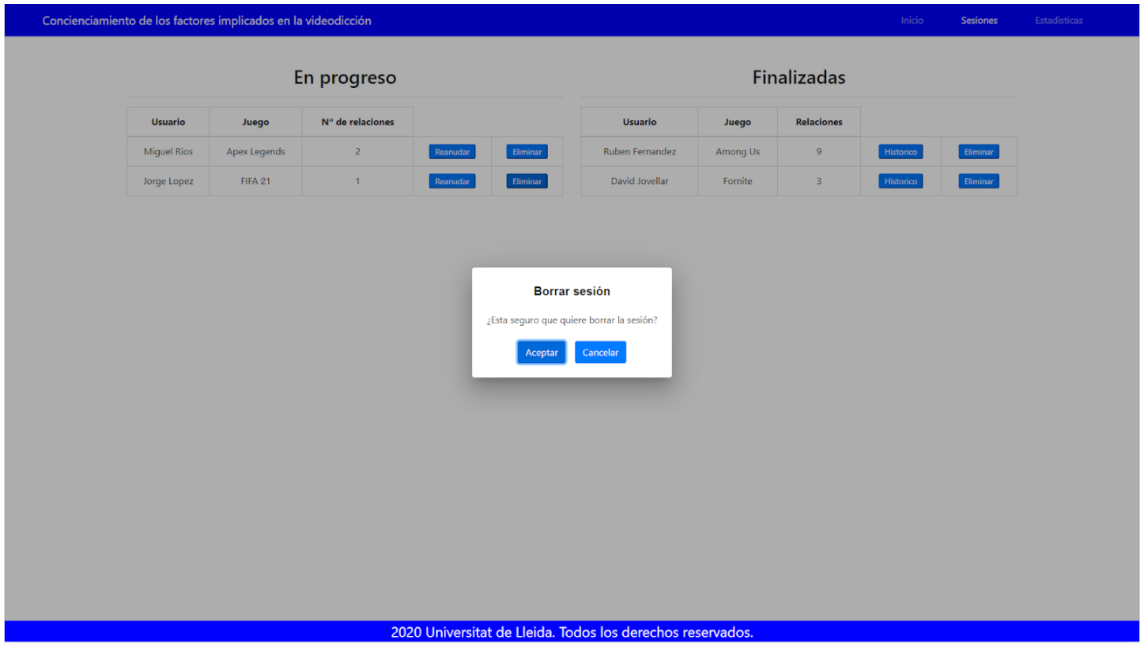


*Figura 16. Iteración 6 - Borrar una sesión guardada*

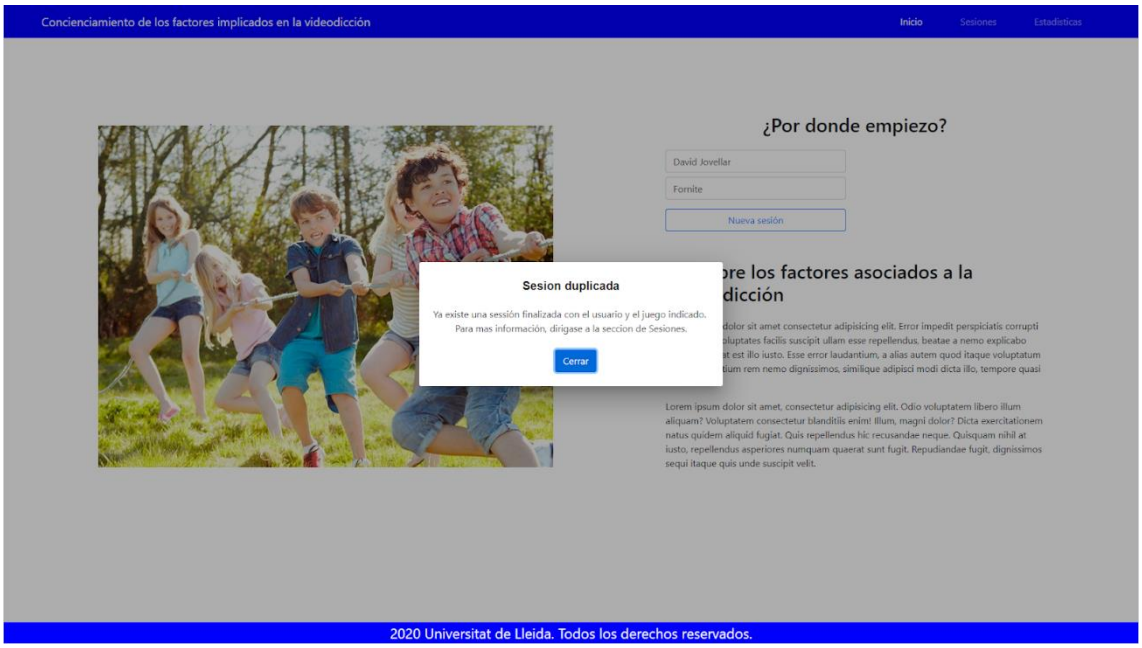


*Figura 17. Iteración 6 – Sesión finalizada duplicada*

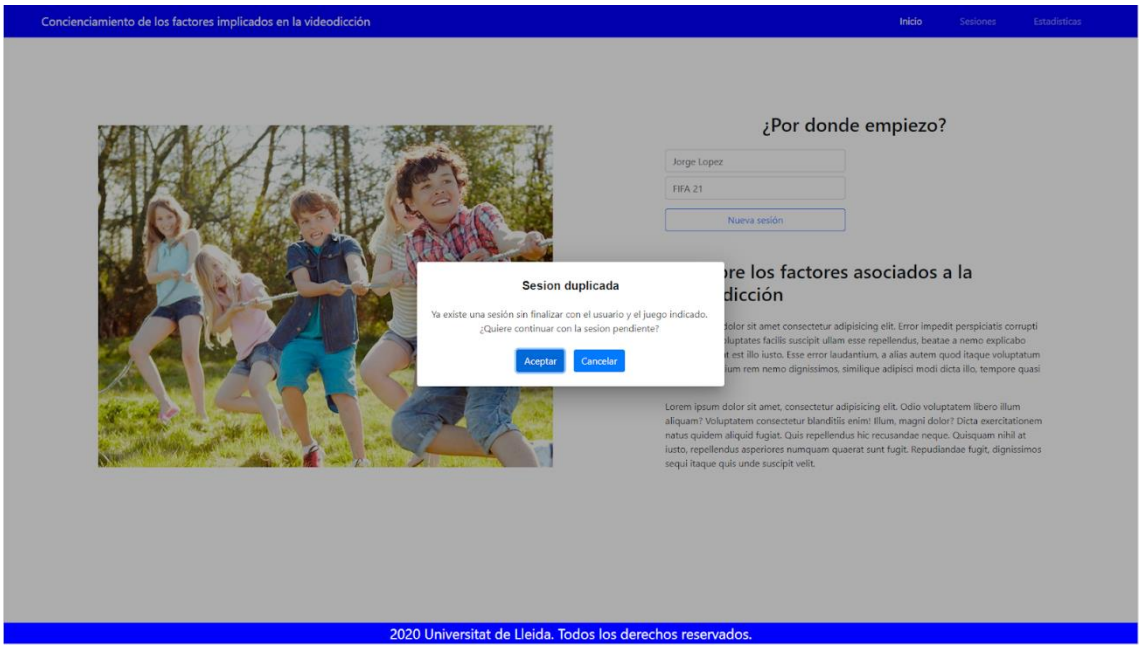


*Figura 18. Iteración 6 - Sesión en progreso duplicada*

### *3.4.6.3 Implementación*

***Guardar múltiples sesiones en memoria simultáneamente***

Para lograr este objetivo se ha modificado la implementación encargada de guardar una sesión en memoria.

Inicialmente, el Servicio Storage se encarga de generar una clave propia para cada sesión a partir del usuario y el juego. Un ejemplo de este formato sería: **Videoadiccio_usuario-juego**.

```
const key = `videoadiccio_${user3}-${game3}`;
```

Al guardar cada sesión con una clave distinta, éstas no se sobrescriben y permiten guardar múltiples sesiones simultáneamente en memoria.

Posteriormente, se ha definido una nueva clase de tipo Modelo encargada de representar toda la información de la sesión. Esta clase se llama SessionInfo y está formada por la clave generada, el usuario, el juego, el histórico de relaciones y una variable de tipo booleana llamada Finish encargada de indicar si la sesión está finalizada.

Finalmente, por cada sesión se crea un objeto de esta nueva clase añadiendo toda la información de la sesión. Para guardar la sesión, se ha definido un método en el Servicio Storage encargado de guardar este objeto serializado usando la clave generada para la sesión.

```
setUser(user: string): void {
  this.sessionInfo.user = user;
}

getGame(): string {
  return this.sessionInfo.game;
}
```

```
saveRelation(relation: RelationCards): void {
  this.sessionInfo.historyRelations.push(relation);
}
```

```
setFinish(finish: boolean): void {
  this.sessionInfo.finish = finish;
}
```

```
this.sessionInfo.key = key;
localStorage.setItem(this.sessionInfo.key,
                                    JSON.stringify(this.sessionInfo));
```

***Página para mostrar las sesiones guardadas***

Se ha creado un nuevo componente de tipo Page encargado de representar la página que muestra todas las sesiones que hay en memoria.

En la página se definen dos apartados encargados de separar las sesiones finalizadas y sin finalizar. Estas sesiones se obtienen mediante el servicio Storage, en el cual se han definido métodos concretos para obtener cada tipo de sesión, diferenciándolas mediante la variable Finish de cada sesión.

```
getPendingSessions(): SessionInfo[] {
  const keys = Object.keys(localStorage);
  const sessions: Array<SessionInfo> = [];

  keys.forEach(key => {
    if (key.startsWith('videoadiccio_')) {
      const session = JSON.parse(localStorage.getItem(key));
      if (!session.finish) {
        sessions.push(session);
      }
    }
  });
  return sessions;
}
```

```
getFinishSessions(): SessionInfo[] {
  const keys = Object.keys(localStorage);
  const sessions: Array<SessionInfo> = [];

  keys.forEach(key => {
    if (key.startsWith('videoadiccio_')) {
      const session = JSON.parse(localStorage.getItem(key));
      if (session.finish) {
        sessions.push(session);
      }
    }
  });
  return sessions;
```

```
}
```

Por cada sesión sin finalizar, se habilita un botón para reanudar la sesión. Esto se consigue creando una nueva sesión añadiendo toda la información de la sesión recuperada.

```
resumeSession(sessionInfo: SessionInfo): void {
  this.storageService.setSessionInfo(sessionInfo);
  this.router.navigate(['session']);
}
```

Por cada sesión finalizada, se habilita un botón para consultar el histórico de relaciones realizado. Esto se consigue cargando el componente que representa el histórico y pasándole como argumento la información de la sesión recuperada.

```
viewHistory(sessionInfo: SessionInfo): void {
  this.storageService.setSessionInfo(sessionInfo);
  this.router.navigate(['session/history']);
}
```

Finalmente, para todas las sesiones se habilita un botón que permite eliminarlas. Esto se consigue borrando la sesión de memoria mediante la clave generada para esa sesión.

```
deleteSession(sessionInfo: SessionInfo, flag: boolean): void {

  this.storageService.deleteSession(sessionInfo.key);

  if (flag) {
    const index = this.pendingSessions.indexOf(sessionInfo);
    this.pendingSessions.splice(index, 1);
  } else {
    const index = this.finishSessions.indexOf(sessionInfo);
    this.finishSessions.splice(index, 1);
  }

}
```

Al intentar borrar una sesión, se muestra una ventana emergente pidiendo confirmación por parte del usuario. Para implementar la ventana emergente se ha utilizado el componente parametrizable implementado anteriormente, cambiando el texto a mostrar y las acciones de los botones.

```
deleteDialog(sessionInfo: SessionInfo, flag: boolean): void {
    const dialogRef = this.dialog.open(DialogBoxComponent, {
    disableClose: true,
    data: {
      title: 'Borrar sesión',
      message: '¿Esta seguro que quiere borrar la sesión?',
      okButton: 'Aceptar',
      noOkButton: 'Cancelar'
    }
    });

    dialogRef.afterClosed().subscribe(res => {
      if (res) {
        this.deleteSession(sessionInfo, flag);
      }
    });
}
```

***<u>Control de sesiones duplicadas</u>***

Al intentar crear una nueva sesión se genera la clave correspondiente y se consulta en caché si existe algún objeto asociado a ella. Si existe una sesión asociada a esta clave, se obtiene de caché y se consulta mediante la variable Finish si está finalizada.

Si la sesión está finalizada, se muestra una ventana emergente informando al usuario y no se crea la sesión. Si la sesión está sin finalizar, se muestra una ventana emergente informando al usuario y dando la opción de reanudarla, creando una nueva sesión con toda la información de la sesión obtenida de caché.

```
const item = localStorage.getItem(key);

if (item) {
  const session = JSON.parse(item);
  if (session.finish) {
    return 'Finish';
  } else {
    return 'Pending';
  }
```

Finalmente, para mejorar el control de duplicados al generar la clave se convierte el usuario y el juego a letras minúsculas y se eliminan todos los espacios en blanco.

```
const user2 = this.sessionInfo.user.replace(/\s/g, '');
const game2 = this.sessionInfo.game.replace(/\s/g, '');
const user3 = user2.toLowerCase();
const game3 = game2.toLowerCase();

const key = `videoadiccio_${user3}-${game3}`;
```

#### *3.4.6.4 Resultados / Conclusiones*

Tras esta iteración se ha completado la funcionalidad encargada de guardar todas las sesiones realizadas, permitiendo posponer sesiones sin finalizar o revisar sesiones finalizadas anteriormente.

Esto permite abrir posibilidades futuras para implementar funcionalidades encargadas de exportar datos y generar estadísticas con todas las sesiones realizadas en la aplicación.

### 3.4.7 Iteración 7

#### *3.4.7.1 Requerimientos / Especificación*

***Borrar relaciones realizadas en la sesión***

Se habilitará un botón en el histórico de relaciones que permita borrar relaciones realizadas durante la sesión.

***Mejorar la presentación visual de la aplicación***

Se modificará el diseño de las pantallas de la aplicación, con el objetivo de mejorar la presentación visual de la aplicación, basándose en la siguiente plantilla:

https://demo.dashboardpack.com/architectui-html-free/

***Rediseño de la pantalla Home***

Además de los estilos indicados en el requerimiento anterior, se convertirá la imagen mostrada en una presentación deslizante automática de imágenes y se añadirá la siguiente descripción de la aplicación:

Según el Instituto Superior de Estudios Psicológicos cada vez son más los niños y jóvenes que juegan a videojuegos y que, como consecuencia de su uso incorrecto, acaban convirtiéndose en adictos a ellos.
Tanto es así, que en 2018, debido a esta creciente problemática, la Organización Mundial de la Salud (OMS), catalogó la adicción a los videojuegos como un desorden de salud mental, incluyéndolo dentro del apartado relacionado con los desórdenes adictivos de la Clasificación Internacional de Enfermedades.
Esta aplicación permite reflexionar sobre los mecanismos diseñados para engancharnos y ver qué riesgos se pueden correr, de forma que se puedan encontrar mecanismos para devolver el control.

### *3.4.7.2 Diseño*

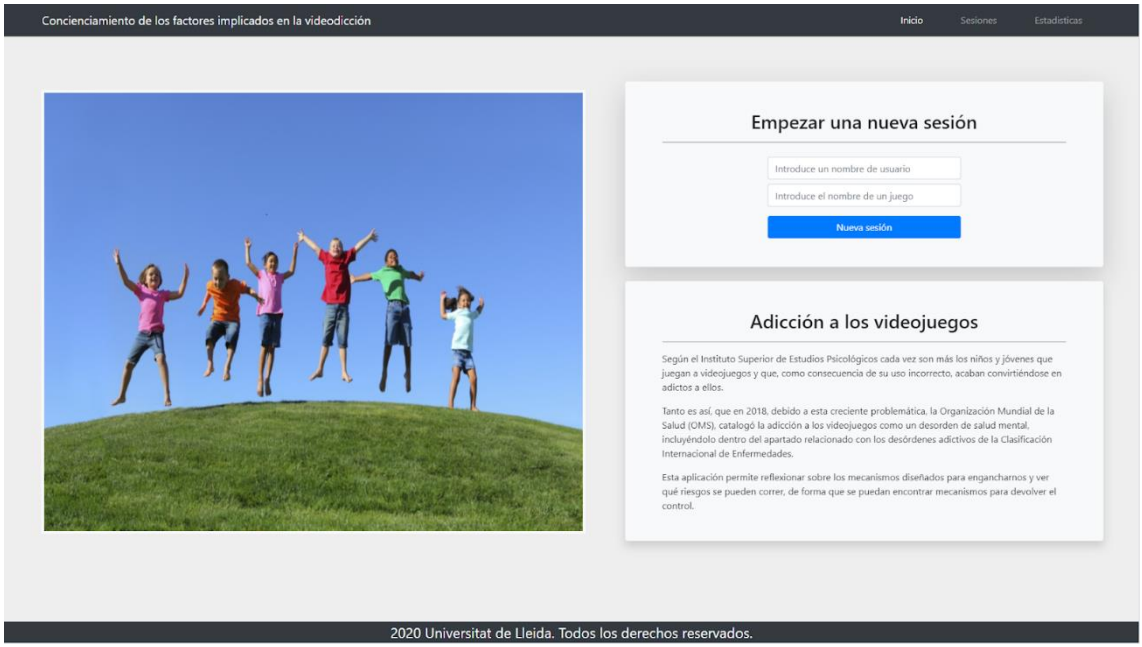


*Figura 19. Iteración 7 - Página Home*

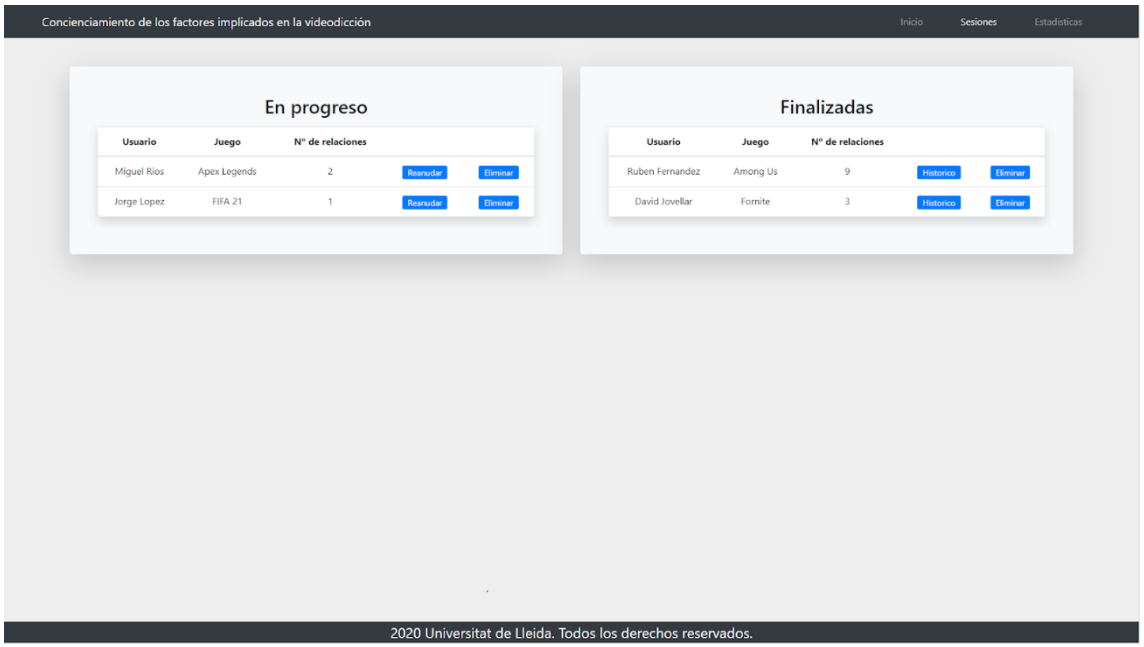


*Figura 20. Iteración 7 - Página de sesiones guardadas*

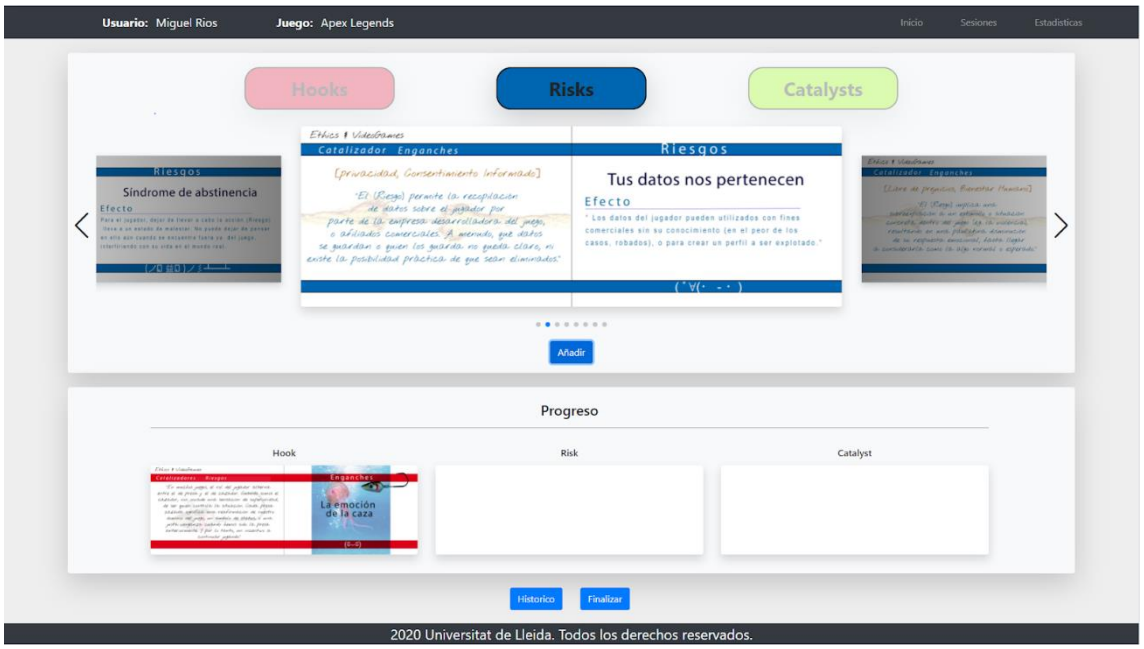


*Figura 21. Iteración 7 - Página de sesión*

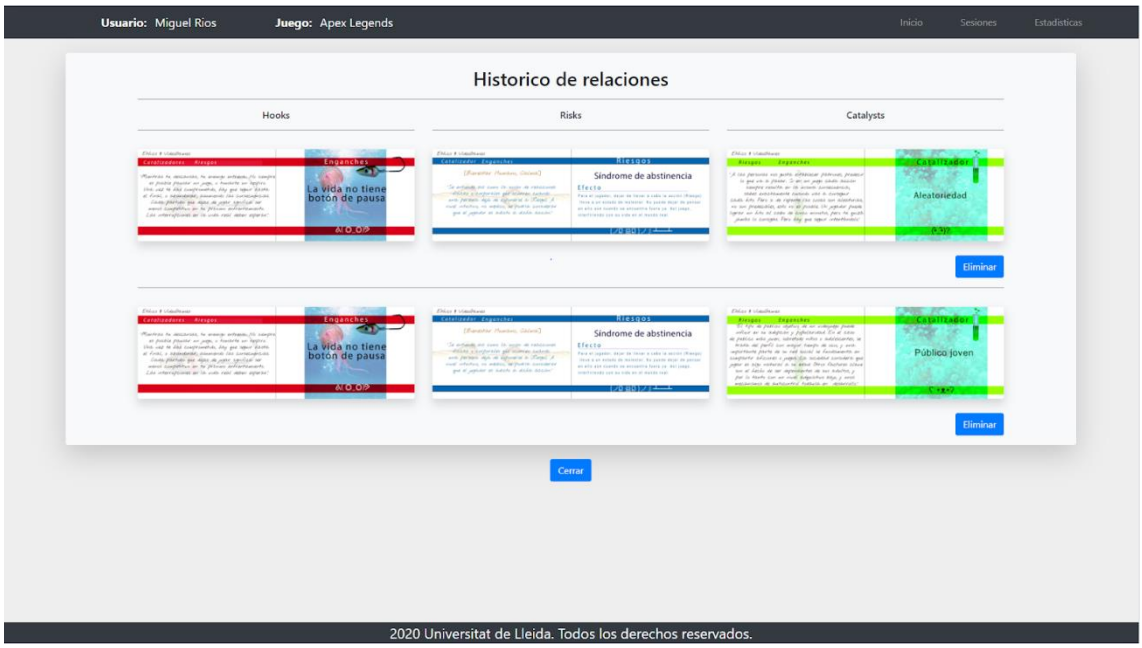


*Figura 22. Iteración 7 - Página de histórico de relaciones*

### *3.4.7.3 Implementación*

***Borrar relaciones realizadas en la sesión***

Para borrar una relación se ha habilitado un botón encargado de borrar la relación concreta de la lista de relaciones del componente History Relations y actualizar la información de la sesión en caché mediante el Servicio Storage.

***Mejorar la presentación visual de la aplicación***

Para diferenciar las distintas secciones de cada página se ha utilizado el componente Shadow de Bootstrap, el cual se encarga de bordear y sombrear la sección indicada (Bootstrap, 2021d).

Siguiendo con el diseño de la plantilla facilitada, se ha definido un color de fondo común para toda la aplicación. Posteriormente, se ha sombreado cada sección de cada página con diferentes tonalidades de blanco/gris para mejorar visualmente cada sección y sus contenidos.

***Rediseño de la pantalla Home***

Para implementar la presentación deslizante de imágenes se ha utilizado el componente de Bootstrap llamado Carousel. Este componente permite crear dicha presentación automática asignándole las imágenes a mostrar (Bootstrap, 2021b).

```
<div id="carouselExampleSlidesOnly"
      class="carousel slide"
      data-ride="carousel">
     <div class="carousel-inner">
         <div class="carousel-item active">
             <img src="assets/img/portada.jpg"
                  class="img-fluid size-img">
         </div>
         <div class="carousel-item">
             <img src="assets/img/portada2.jpg"
                  class="img-fluid size-img">
         </div>
         <div class="carousel-item">
             <img src="assets/img/portada3.jpg"
                  class="img-fluid size-img">
         </div>
         <div class="carousel-item">
             <img src="assets/img/portada4.jpg"
                  class="img-fluid size-img">
         </div>
         <div class="carousel-item">
             <img src="assets/img/portada5.jpg"
                  class="img-fluid size-img">
         </div>
     </div>
 </div>
```

Finalmente, se ha modificado el texto ficticio de la descripción por el texto real proporcionado y se han aplicado los estilos descritos en el requerimiento anterior.

#### *3.4.7.4 Resultados / Conclusiones*

Tras esta iteración se ha mejorado sustancialmente la presentación visual de la aplicación, delimitando claramente cada sección de cada página mediante el bordeado y el sombreado.

### 3.4.8 Iteración 8

#### *3.4.8.1 Requerimientos / Especificación*

***Exportar la información de las sesiones finalizadas***

Con el objetivo de poder realizar análisis y estadísticas externamente, se habilitará un botón que permite descargar localmente un Excel con toda la información de las sesiones finalizadas.

***Enviar información de sesión por correo electrónico***

Con el objetivo de tener una herramienta de guardado en remoto, se habilitará un botón por sesión finalizada para enviar la información de la sesión por correo electrónico.

#### *3.4.8.2 Diseño*

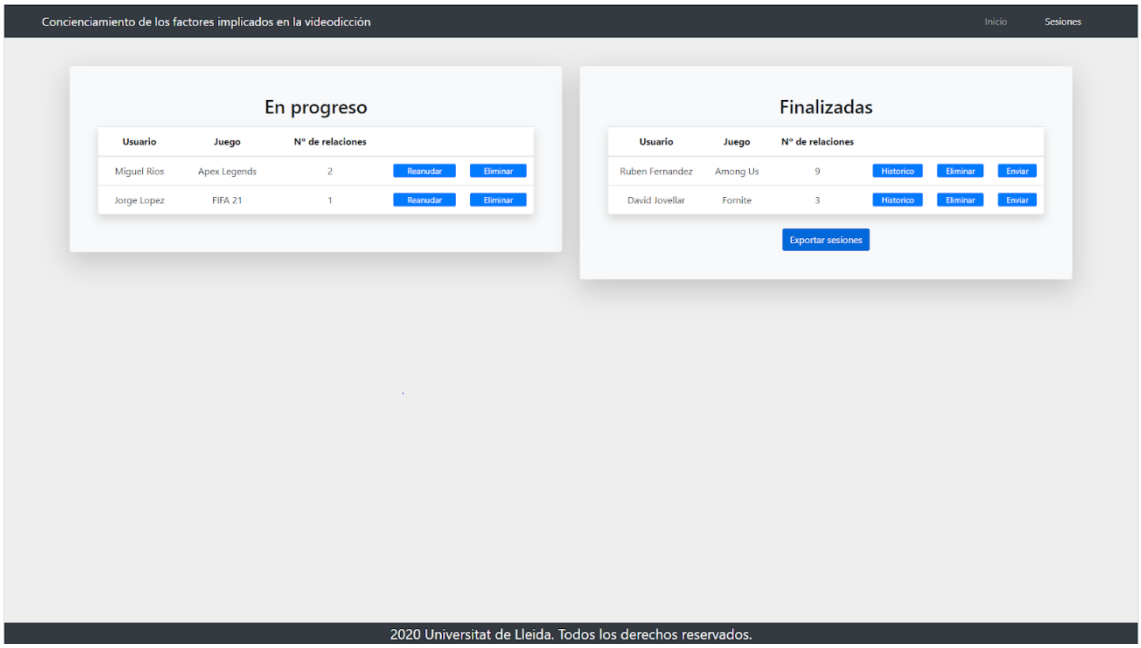


*Figura 23. Iteración 8 - Exportar sesiones finalizadas*

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **Usuario** | **Juego** | **Hook** | **Risk** | **Catalyst** | | |
| 2 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Es la moda | | |
| 3 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Microtransacciones | | |
| 4 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Microtransacciones | | |
| 5 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 6 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 7 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 8 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 9 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 10 | Ruben Fernandez | Among Us | La emocion de la caza | Refuerzo de los prejuicios | Aleatoriedad | | |
| 11 | David Jovellar | Fornite | La vida no tiene boton de pausa | Abstinencia | Aleatoriedad | | |
| 12 | David Jovellar | Fornite | La vida no tiene boton de pausa | Abstinencia | Aleatoriedad | | |
| 13 | David Jovellar | Fornite | La vida no tiene boton de pausa | Abstinencia | Aleatoriedad | | |
| 14 | | | | | | | |
| 15 | | | | | | | |
| 16 | | | | | | | |
| 17 | | | | | | | |
| 18 | | | | | | | |
| 19 | | | | | | | |
| 20 | | | | | | | |
| 21 | | | | | | | |
| 22 | | | | | | | |
| 23 | | | | | | | |
| 24 | | | | | | | |
| 25 | | | | | | | |
| 26 | | | | | | | |
| 27 | | | | | | | |
| 28 | | | | | | | |
| 29 | | | | | | | |
| 30 | | | | | | | |
| 31 | | | | | | | |

*Figura 24. Iteración 8 - Excel generado al exportar las sesiones finalizadas*

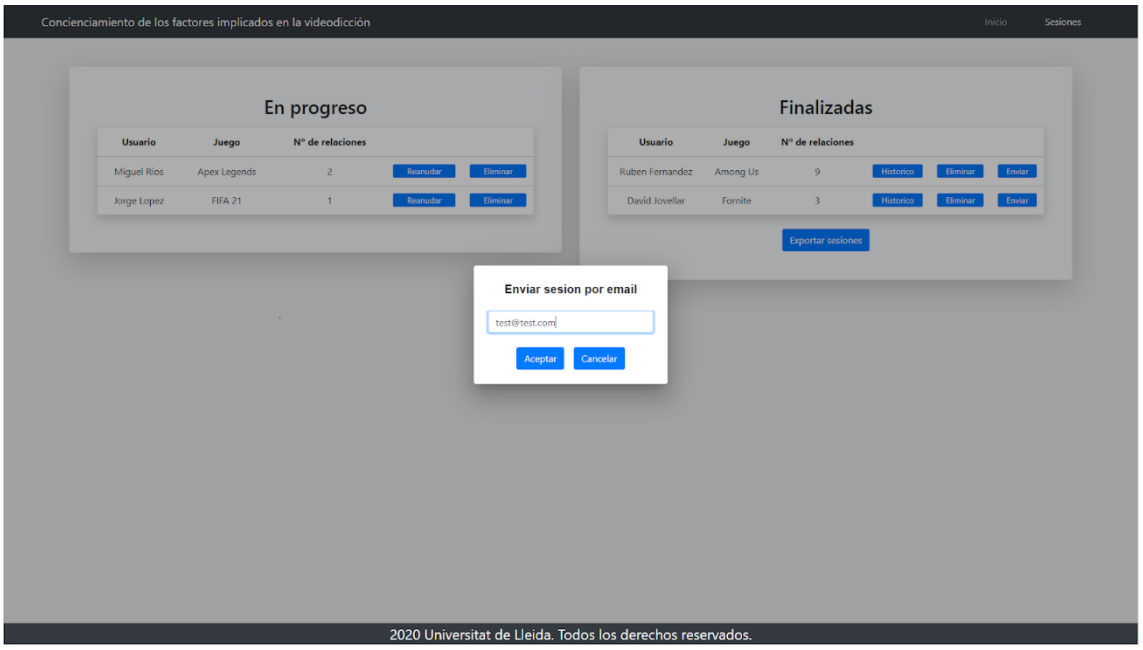


*Figura 25. Iteración 8 – Solicitar remitente para enviar sesión*

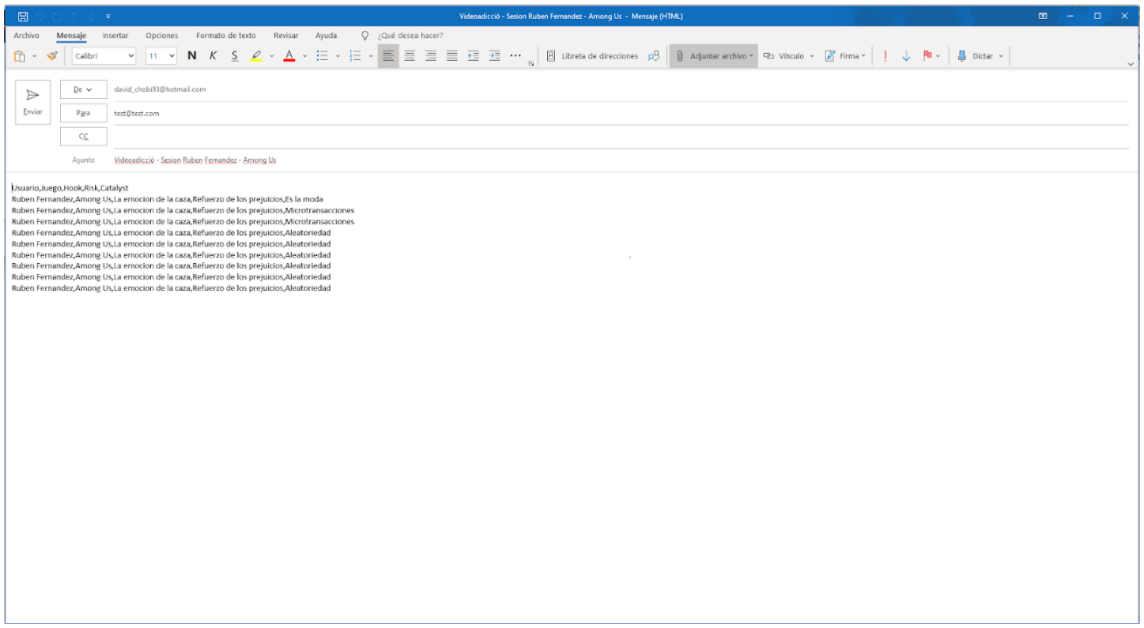


*Figura 26. Iteración 8 - Enviar sesión finalizada por correo*

### *3.4.8.3 Implementación*

***Exportar la información de las sesiones finalizadas***

Se ha creado un Servicio llamado Export encargado de realizar la lógica que generará el fichero Excel a descargar. Este servicio hace uso de la librería de Angular Excel JS, la cual permite crear hojas de estilo con un alto grado de personalización (Prajapati, 2020).

El formato del Excel se compone de 5 columnas que representan el Usuario, Juego, Tarjeta Hook, Tarjeta Risk y Tarjeta Catalyst. A continuación se añaden tantas filas como relaciones se han realizado entre todas las sesiones finalizadas.

```
const workbook = new Workbook();

const worksheet = workbook.addWorksheet('Sesiones');

const header = worksheet.addRow(['Usuario', 'Juego',
                                 'Hook', 'Risk', 'Catalyst']);
header.font = { size: 12 , bold: true, color: { argb: 'FFFFFFFF' } };
header.alignment = { vertical: 'middle',
                     horizontal: 'center', wrapText: true };

header.eachCell((cell) => {
  cell.fill = {
    type: 'pattern',
    pattern: 'solid',
    fgColor: { argb: 'FF000000' },
    bgColor: { argb: 'FF0000FF' }
  };
});
```

```
worksheet.columns.forEach( column => {
  column.width = 40;
});

sessions.forEach(session => {
  session.historyRelations.forEach(relation => {
    const content = worksheet.addRow(
      [session.user, session.game, relation.hook.name,
           relation.risc.name, relation.catalyst.name]
    );
    content.alignment = { vertical: 'middle',
                          horizontal: 'center', wrapText: true };
  });
});
```

Finalmente, al terminar la construcción del fichero el Servicio Storage utiliza el Módulo de Angular File-saver para descargarlo localmente de forma automática (NPM, 2020).

```
workbook.xlsx.writeBuffer().then(
  data => {
    const blob = new Blob(
      [data], { type: 'application/vnd.openxmlformats-
                          officedocument.spreadsheetml.sheet' }
    );
    fs.saveAs(blob, 'data-sessions.xlsx');
  }
);
```

***Enviar información de sesión por correo electrónico***

Dado que la aplicación no dispone de un Backend no es posible enviar un correo electrónico de forma segura. Para salvar esta limitación se ha utilizado la herramienta <mailto>, la cual permite preparar el envío de un correo electrónico indicando el destinatario, el asunto y el cuerpo del mensaje mediante una URL (Malek, 2020).

Para generar esta URL y abrirla en el navegador se ha utilizado el Servicio Export, el cual se encarga de obtener toda la información necesaria. El destinatario se le solicita al usuario mediante una ventana emergente, en cambio el asunto y el cuerpo del mensaje se encarga de generarlo el propio servicio.

En el cuerpo del mensaje es donde se inserta la información de la sesión (usuario, juego y relaciones realizadas), la cual previamente se encarga el Servicio de serializar.

```
sendSessionByEmail(session: SessionInfo, email: string): boolean {

  let body = `Usuario,Juego,Hook,Risk,Catalyst`;

  session.historyRelations.forEach( relation => {
    body += `%0D%0A${session.user},${session.game},
                    ${relation.hook.name},${relation.risc.name},
                          ${relation.catalyst.name}`;
  });

  const mailto = `mailto:${email}?subject=Videoadicció - Sesion
                  ${session.user} - ${session.game}&body=${body}`;

  if (mailto.length > 1750) {
    return false;
  } else {
    window.open(mailto, '_blank');
    return true;
  }

}
```

Para poder trabajar con la información serializada se debe copiar la información del cuerpo del mensaje en un fichero de extensión TXT o CSV y abrirlo mediante Excel. De esta forma se obtiene un fichero casi idéntico (se pierde algún formato) a la hoja de estilo generada localmente (Microsoft, 2021).

#### *3.4.8.4 Resultados / Conclusiones*

Gracias a esta iteración se ha implementado la posibilidad de extraer la información generada al realizar las sesiones. Se brinda la posibilidad de generar un fichero y exportarlo localmente o de exportar por correo electrónico la información de cada sesión individualmente.

Esta segunda opción se ha implementado como última opción, para no obligar al usuario a tener que exportar la información forzosamente de forma local.

Mediante esta información se da la posibilidad a los usuarios de poder realizar todo tipo de estadísticas externamente, lo cual puede ser sumamente importante dado que el principal objetivo de la aplicación es extraer conclusiones sobre la videoadicción.

# 4. Conclusiones

Cuando empecé este proyecto no estaba seguro de si sería capaz de sobrellevar toda la carga de trabajo que supondría, ya que debía compaginarlo con mi trabajo como desarrollador a jornada completa.
Aunque al principio fue un poco estresante, poco a poco fui aprendiendo a compaginar ambas cargas de trabajo y optimizar el tiempo disponible que disponía para dedicar a este proyecto.

Tras varios meses de trabajo, estoy a punto de finalizar el proyecto y puedo concluir que he aprendido que, con una buena planificación y aprovechando al máximo cada hora del día, soy capaz de compaginar grandes cargas de trabajo.

Respecto al aprendizaje autodidacta previo al inicio del desarrollo, me he dado cuenta de que me centré demasiado en la parte lógica de Angular, es decir, el lenguaje Typescript y todas las posibilidades al desarrollar la lógica en el lado del controlador. Esto provocó que no dedicara el suficiente tiempo a aprender todas las posibilidades al desarrollar el lado de la vista, es decir, los lenguajes HTML, CSS o el Framework Bootstrap. Debido a esto, las partes más lentas del desarrollo han sido en el momento de diseñar las vistas de la aplicación.

Finalmente, una de las principales motivaciones al comenzar este proyecto era que la aplicación desarrollada pudiera aportar valor y ser de utilidad. Esto se pretendía conseguir facilitando la comprensión del proceso de videoadicción en las personas. Tras finalizar el desarrollo opino que, aunque aún hay margen de mejora, la aplicación desarrollada cumple perfectamente estos propósitos.

# 5. Trabajo Futuro

Debido al tiempo disponible hay algunas ideas sobre funcionalidades que se han quedado fuera del desarrollo. A continuación se van a detallar las que se consideran más importantes.

## 5.1 Desarrollar una parte Backend

Uno de los aspectos más importantes que se ha quedado fuera del desarrollo ha sido implementar una parte Backend con la que la parte Frontend pudiera comunicarse. Esto abriría grandes posibilidades a implementar funcionalidades muy provechosas.

Gran parte de las siguientes funcionalidades que se van a detallar no han sido posibles (o son posibles pero más complejas a la vez que menos consistentes) debido a la falta de una parte Backend.

## 5.2 Enviar la información de la sesión por email sin serializar

Debido a la no existencia de una parte Backend de la aplicación se ha tenido que utilizar, como alternativa, el envío por email de la información de cada sesión usando la funcionalidad proporcionada por Mailto (Malek, 2020).

Esta funcionalidad tiene mucho margen de mejora, ya que depende de un cliente de correo electrónico en la máquina en la que se ejecuta e implica que el usuario debe copiar la información enviada a un fichero TXT o CSV e introducirla en Excel para que la procese (Microsoft, 2021).

Con una parte Backend el propio servidor podría utilizar el remitente para enviar él mismo el correo electrónico adjuntando el propio fichero Excel con la información de todas las sesiones (o generando uno nuevo solo de esa sesión siguiendo la misma implementación) que genera la parte Frontend.

De esta forma, se reduciría mucho la complejidad de esta funcionalidad.

## 5.3 Registro y autenticación de usuarios

Aprovechando la implementación de la parte Backend se podría implementar una funcionalidad que permitiera registrar usuarios, posibilitando de esta forma añadir un alto grado de personalización de las sesiones por parte de cada usuario.

## 5.4 Tarjetas personalizadas en la aplicación

Con una parte Backend y un registro de usuarios, se podría crear una pantalla de configuración permitiendo al usuario personalizar de distintas formas las sesiones que se crearan.

Dado que las tarjetas se definen mediante un fichero JSON que contiene la información de cada tarjeta a crear, se podría permitir modificar este fichero añadiendo nuevas tarjetas, eliminarlas o modificar el nombre, la imagen o el tipo de las tarjetas ya existentes.

Incluso se podría permitir sustituir este fichero JSON por otro creado por el propio usuario, validando previamente que cumpla ciertos requisitos para asegurar el correcto funcionamiento de la aplicación.

# 6. Bibliografía


Atlassian. (2021). *Flujo de trabajo de Gitflow | Atlassian Git Tutorial*. Atlassian. https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow

Bootstrap. (2021a). *Alerts · Bootstrap*. Getbotstrap. https://getbootstrap.com/docs/4.0/components/alerts/

Bootstrap. (2021b). *Carousel · Bootstrap*. Getbootstrap. https://getbootstrap.com/docs/4.1/components/carousel/

Bootstrap. (2021c). *Grid system · Bootstrap*. Getbootstrap. https://getbootstrap.com/docs/4.0/layout/grid/

Bootstrap. (2021d). *Shadows · Bootstrap*. Getbootstrap. https://getbootstrap.com/docs/4.1/utilities/shadows/

Colina, C. (2019). *¿Cuáles son las principales ventajas de TypeScript sobre JavaScript (ES6, 7, 8)? - Quora*. Quora. https://es.quora.com/Cuáles-son-las-principales-ventajas-de-TypeScript-sobre-JavaScript-ES6-7-8

Gil, R. M., & Arnedo-moreno, J. (2020). Designing an activity to help reflect on " Healthy Engagement vs Video Game Designing an activity to help reflect on " Healthy Engagement vs Video Game Addiction ". *TEEM 2020*, *October*.

Google. (2020a). *Angular - Reactive forms*. Angular. https://angular.io/guide/reactive-forms

Google. (2020b). *Dialog | Angular Material*. Angular Material. https://material.angular.io/components/dialog/overview

Google. (2020c). *Drag and Drop | Angular Material*. Angular Material. https://material.angular.io/cdk/drag-drop/overview

Kharlampidi, V. (2020). *Swiper Angular*. SwiperJS. https://swiperjs.com/angular/

Malek, P. (2020). *Mailto Links Explained with Examples | Mailtrap Blog*. Mailtrap. https://blog.mailtrap.io/mailto-links-explained/

Microsoft. (2021). *Importar o exportar archivos de texto (.txt o .csv) - Excel*. Support Microsoft. https://support.microsoft.com/es-es/office/importar-o-exportar-archivos-de-texto-txt-o-csv-5250ac4c-663c-47ce-937b-339e391393ba

NPM. (2020). *file-saver - npm*. NPM. https://www.npmjs.com/package/file-saver

Palomares, K. (2019). *¿Qué es una web SPA? - Single Page Application*. Kikopalomares. https://www.kikopalomares.com/blog/que-es-una-web-spa-single-page-application/

Prajapati, A. (2020). *Export to Excel in Angular 8 using ExcelJS*. Ng Develop. https://www.ngdevelop.tech/export-to-excel-in-angular-6/

Quality devs. (2019). *¿Qué es Angular y para qué sirve?* Quality devs. https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/

Rodríguez Patiño, E. (2021). *¿Qué son las single-page application (SPA)?, ventajas y desventajas*. Anexsoft. https://anexsoft.com/que-son-las-single-page-application-spa-ventajas-y-desventajas

Suárez, D. (2020). *Bootstrap 4: Qué es, cómo instalarlo en tu web y cómo se utiliza*. Raiolanetworks. https://raiolanetworks.es/blog/bootstrap/

*TypeScript: Typed JavaScript at Any Scale.* (2021). https://www.typescriptlang.org/